МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний технічний університет
«Харківський політехнічний інститут»

МЕТОДИЧНІ ВКАЗІВКИ
до лабораторної роботи
«Створення консольного застосування у C++Builder 2009»

з курсу «Програмування»
для студентів напряму 6.040302 – Інформатика
і курсу «Програмування та алгоритмічні мови»
для студентів напряму 6.040303 – Системний аналіз

Затверджено редакційно-видавничою
радою університету,
протокол № 3 від 28.12.09.

Харків НТУ «ХПІ» 2010

Методичні вказівки до лабораторної роботи «Створення консольного застосування у C++Builder 2009» з курсу «Програмування» для студентів напряму 6.040302 – Інформатика і курсу «Програмування та алгоритмічні мови» для студентів напряму 6.040303 – Системний аналіз / Уклад. М. І. Безменов, І. І. Марченко. – Х. : НТУ «ХПІ», 2010. – 28 с.

Укладачі: М. І. Безменов,
І. І. Марченко

Рецензент Л. М. Любчик

Кафедра системного аналізу і управління

Мета роботи

Освоєння середовища розробки Code Gear C++ Builder 2009.

1. ТЕОРЕТИЧНІ ОСНОВИ

1.1. Початкові дії

Після завантаження інтегрованого середовища потрібно здійснити перехід до розробки нового консольного застосування. Порядок дій при цьому може бути таким:

1. Вибрати у головному меню опцію File, наслідком чого буде розкриття підменю, першою опцією якого є опція New.
2. У розкритому меню вибрати опцію New, в результаті чого відкривається підменю, однією з опцій якого є опція Other... (Інший...).
3. Вибрати опцію Other..., що приведе до відкриття вікна New Items (див. рис. 1.1).

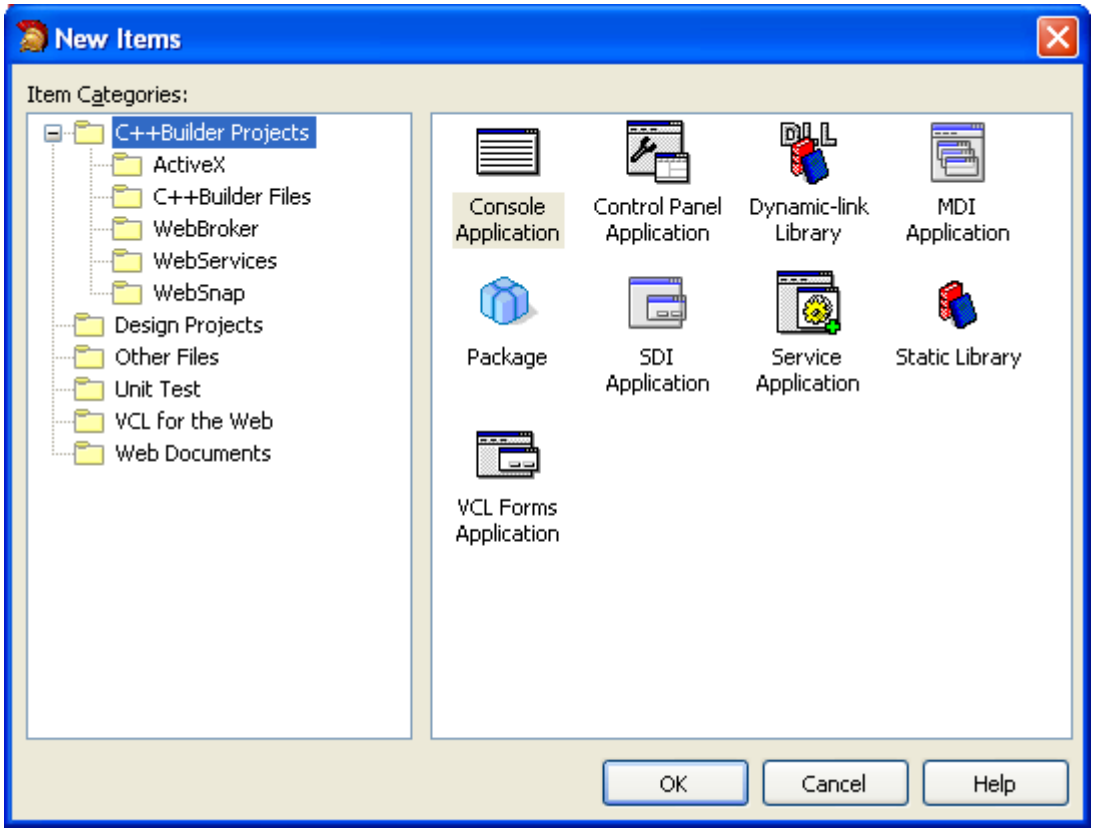

Рис. 1.1. Вікно вибору виду додатка, що проектуватиметься

4. У вікні New Items вибрати пункт Console Application, наслідком чого буде відкриття вікна New Console Application, зображення якого наведено на рис. 1.2.

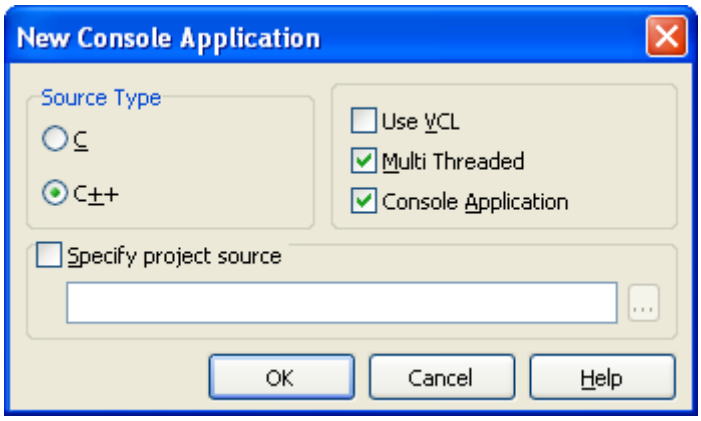

Рис. 1.2. Вікно налаштування нового консольного застосування

5. Здійснити налаштування у вікні New Console Application та клікнути мишкою над його кнопкою Ok.

Послідовність дій пунктів 1–4 можна позначити так:

File ►New ►Other... ►Console Application

Примітка. Можна одразу увійти у вікно New Items, що відкривається за File ►New. Для цього треба клікнути мишкою над інструментальною кнопкою  (New Items).

У цьому вікні можна виконати такі налаштування:

1. Вибрати мову програмування (C або C++), синтаксис якої буде використовуватися у розроблюваному головному модулі. Для цього треба активізувати відповідний перемикач у групі перемикачів Source Type (звісно, треба увімкнути перемикач C++).
2. Дозволити або заборонити використання у проекті бібліотеки візуальних компонентів (Visual Component Library – VCL). Якщо встановити прапорець Use VCL, то ресурси VCL-бібліотеки (насамперед, безпосередньо візуальні компоненти) можуть бути використані навіть у консольному застосуванні. Ця опція є доступною тільки у разі обирання синтаксису мови C++ у групі Source Type (у мові C відсутні класи як такі, а отже, і компоненти).
3. Дозволити створення мультипотокове (багатопотокове) застосування, для чого повинен бути встановлений прапорець Multi Threaded.

Оскільки підтримка мультипотоків входить у склад бібліотеки VCL, у разі встановлення прапорця Use VCL опція Multi Threaded стає недоступною.

4. Безпосередньо вказати на створення консольного застосування, для чого треба встановити (залишити встановленим) прапорець Console Application.
5. Якщо існує cpp-файл (або c-файл), що може бути початковим для створення консольного застосування, то його можна завантажити, встановивши прапорець Specify project source і вказавши ім'я цього файлу у однорядковому редакторі, розміщеному нижче цього прапорця або вибрати файл, скориставшись кнопкою  , що знаходиться праворуч від вікна цього редактора.

Для збереження вмісту вікна коду треба клікнути мишкою над інструментальною кнопкою  (Save), або обрати пункт меню File ►Save, або натиснути сполучення клавіш Ctrl+S. Якщо раніше збереження коду не здійснювалось, відкриється стандартне вікно збереження файлу, у якому потрібно виконати необхідні дії. У іншому випадку збереження здійсниться без попереджень.

У разі потреби збереження коду у файлі з новим іменем відносно того, з яким він записувався на диск раніше, треба вибрати пункт меню File ►Save As..., результатом чого буде відкриття стандартного вікна збереження файлу.

У C++Builder 2009 консольне застосування функціонує не самостійно, а у складі проекту, який створюється автоматично і звичайно записується на диск разом із файлом коду консольного застосування. Збереження проекту разом з текстом безпосередньо консольного застосування здійснюється обиранням пункту меню File ►Save All або кліком мишкою над інструментальною кнопкою  (можна також натиснути клавішу Shift+Ctrl+S). Проект може бути збережений також після обирання пункту меню File ►Save Project As...

1.2. Робота з вікном коду

Це вікно є звичайним вікном багаторядкового текстового редактора, і саме в ньому здійснюється набирання тексту програми.

Початковий вигляд вікна коду для консольного застосування зображений на рис 1.3. Якщо програма є багатомодульною, кожен модуль може бути

завантажений на окремій сторінці вікна коду, причому кожна сторінка має закладку з іменем модуля (на рис. 1.3 завантажений тільки один файл).

Робота з вікном коду не має ніяких особливостей відносно роботи у будь-якому іншому редакторі:

- набирання тексту здійснюється у позиції, де знаходиться курсор;
- переміщення курсору здійснюється кліком мишкою над потрібним місцем вікна, або за допомогою клавіш керування курсором, а саме:
  - стрілка ліворуч, стрілка праворуч, стрілка вгору, стрілка вниз – переміщення курсору на одну позицію у відповідному напрямі;
  - Home – переміщує курсор на початок поточного рядка;
  - End – переміщує курсор на кінець поточного рядка;
  - Page Up – переміщує курсор на одну сторінку екрану вгору;
  - Page Down – переміщує курсор на одну сторінку екрану вниз;
  - Ctrl+стрілка ліворуч – переміщує курсор на одно слово ліворуч (до найближчого символу-роздільника);
  - Ctrl+стрілка праворуч – переміщує курсор на початок наступного слова (усі символи-роздільники пропускаються);
  - Ctrl+стрілка вгору – переміщує редакторське вікно відносно тексту на один рядок угору (візуально текст у редакторському вікні зсувається на один рядок вниз);
  - Ctrl+стрілка вниз – переміщує редакторське вікно відносно тексту на один рядок вниз (візуально текст у редакторському вікні зсувається на один рядок угору);
  - Ctrl+Home – переміщує курсор на початок тексту;
  - Ctrl+End – переміщує курсор на кінець тексту;
  - Ctrl+Page Up – переміщує курсор на перший рядок тексту у вікні редактора, не змінюючи його положення по горизонталі;
  - Ctrl+Page Down – переміщує курсор на останній рядок тексту у вікні редактора, не змінюючи його положення по горизонталі;

- натискання клавіші Backspace приводить до знищення одного символу безпосередньо перед курсором;
- натискання клавіші Delete приводить до знищення одного символу безпосередньо за курсором;
- клавіша Ctrl+Backspace знищує усі символи поточного слова ліворуч від курсору до найближчого символу-роздільника;
- клавіша Ctrl+Delete знищує усі символи поточного слова праворуч від курсору до найближчого символу-роздільника;
- відновити текст до останнього редагування можна за допомогою пункту меню Edit ►Undo або натисканням клавіші Ctrl+Z (це можна робити декілька разів підряд):
- відновити текст після останнього редагування (у тому числі декілька разів підряд) можна за допомогою пункту меню Edit ►Redo або натисканням клавіші Shift+Ctrl+Z;
- виділення тексту здійснюється зміщенням курсору при натиснутій лівій кнопці мишки або зміщенням курсору за допомогою клавіш керування ним при натиснутій клавіші Shift (можна також натиснути клавішу Shift і клікнути мишкою над позицією, у якій закінчується виділення);
- натискання будь-якої символьної клавіші у разі наявності виділення приводить до заміни виділеного тексту введеним символом;
- натискання клавіш Backspace та Delete приводить до знищення всього виділеного тексту, а не одного символу, як це буває у разі відсутності виділення;
- виділений текст може бути скопійований у буфер обміну, для чого треба вибрати пункт меню Edit ►Copy або натиснути клавішу Ctrl+C (або Ctrl+Insert);
- для копіювання виділеного тексту з його знищенням (вирізання) служить пункт меню Edit ►Cut або клавіша Ctrl+X (або Ctrl+Delete);
- вставка тексту, що знаходиться в буфері обміну, здійснюється вибором пункту меню Edit ►Paste або натисканням клавіші Ctrl+V (або Shift+Insert).

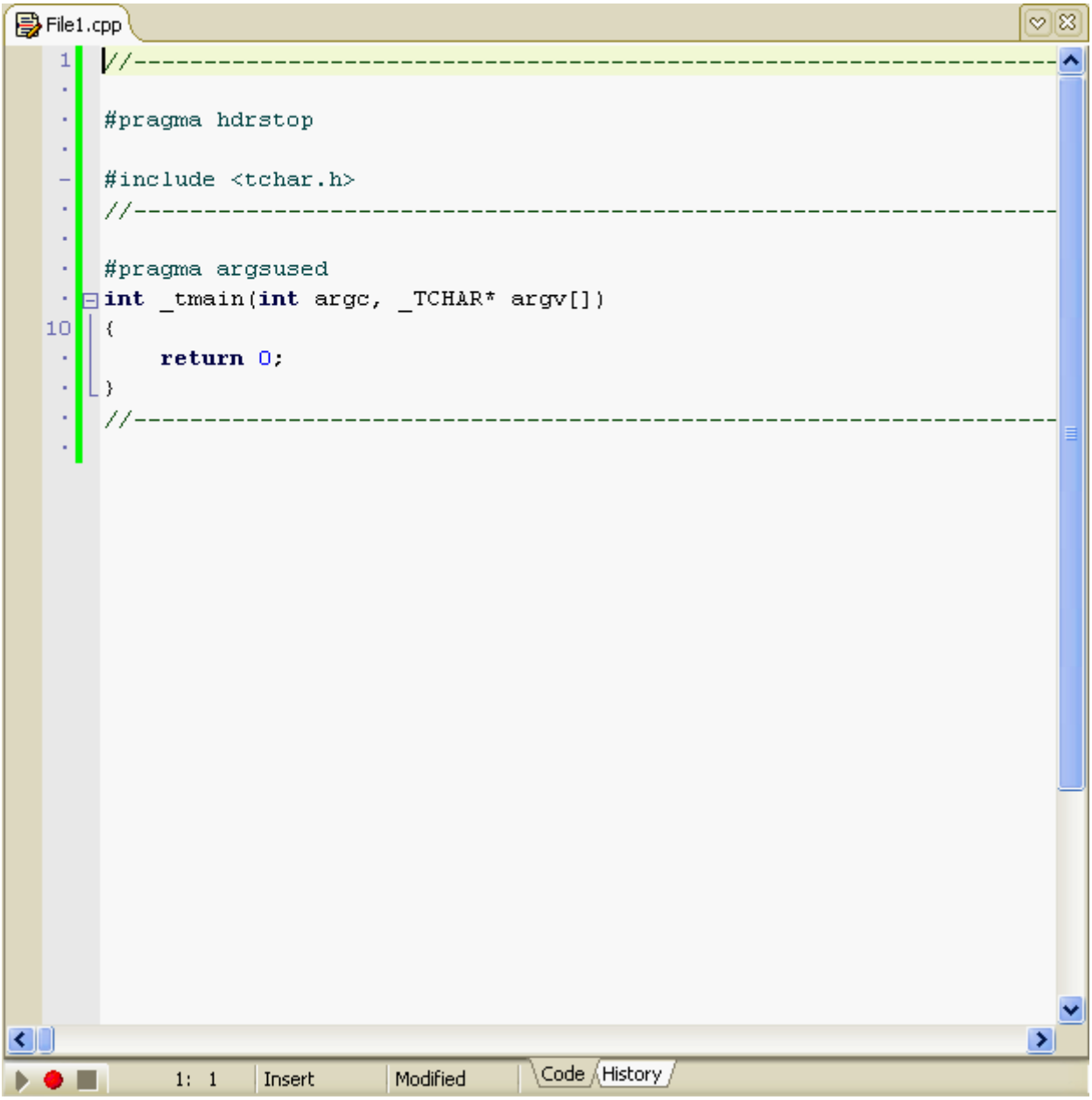

Рис. 1.3. Початковий вигляд вікна коду для консольного додатка

При роботі з редактором коду є можливість перегляду історії змін, що вносилися в код програми. Для цього треба клікнути мишкою над вкладкою History у нижній частині вікна коду, яке при створенні консольного застосування має дві сторінки – сторінку редактора коду (Code) і сторінку історії змін (History). За допомогою останньої забезпечується доступ до версій коду, збереження яких здійснювалось під час його виправлення. Сторінка історії змін дозволяє не тільки переглядати зміни, але й здійснювати повернення до однієї зі збережених версій коду програми (рис. 1.4), причому це можна зробити у будь-який час.

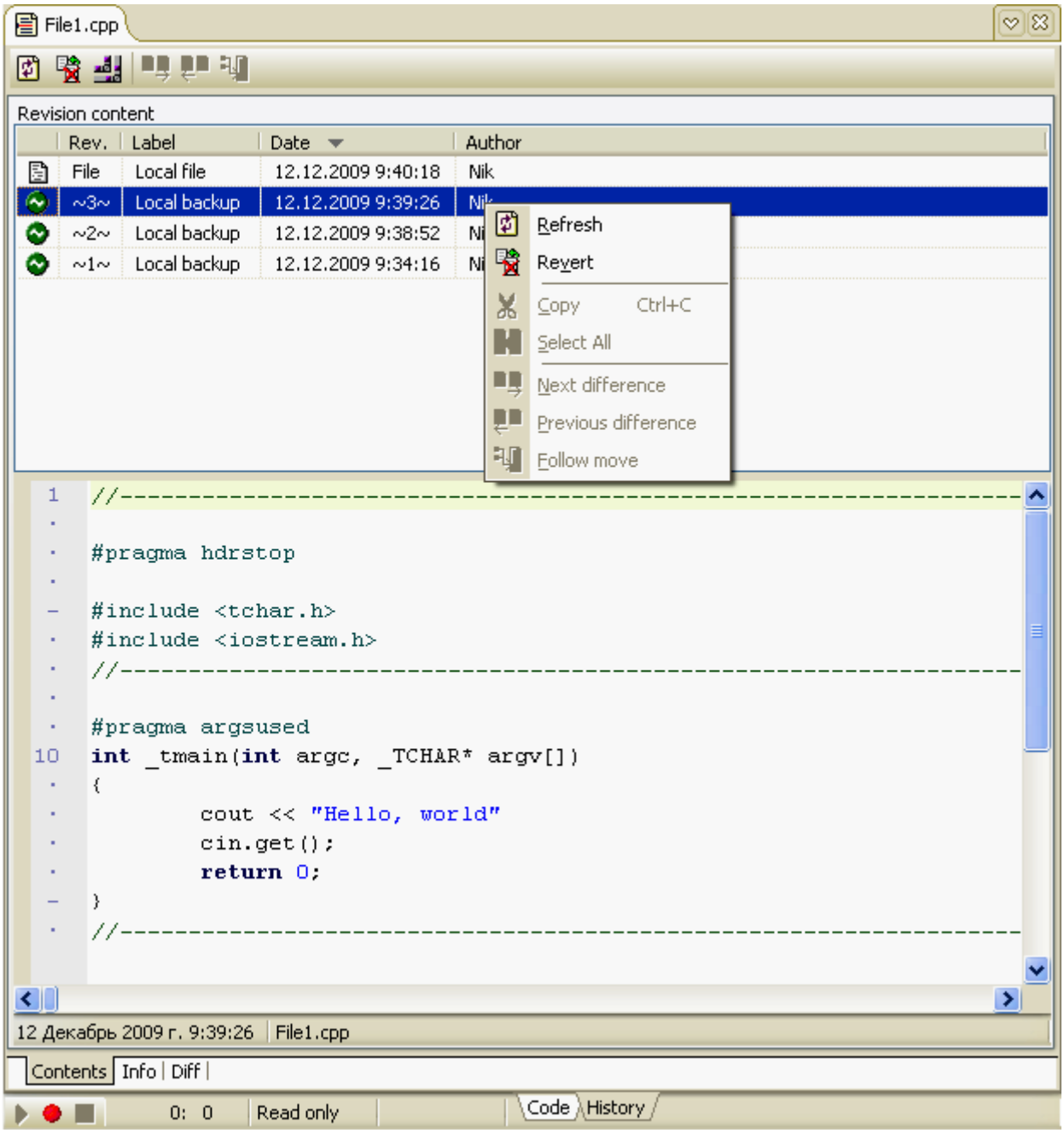

Рис 1.4. Сторінка історії змін

1.3. Компіляція та компонування

Компілятор (Compiler) – це особлива програма, для якої як вхідні дані виступають програми, написані мовою високого рівня, і на виході якої також утворюються програми, але вже написані машинною мовою.

Програми, отримані на виході компілятора, називаються ***об'єктними програмами***, або ***об'єктними кодами***. Особливістю таких програм є те, що

вони не можуть бути відразу виконані, оскільки до того необхідно здійснити підключення до них деяких службових підпрограм.

Остаточне опрацювання програми, точніше об'єктного коду, здійснює так звана ***програма-компонувальник*** (інакше ***редактор зв'язків***, Linker). Саме компонувальник створює ***машинний код***, який і виконується комп'ютером при розв'язанні задачі. Таким чином, компонувальник поєднує об'єктний код програми з об'єктними кодами службових підпрограм, які використовуються даною програмою.

1.4. Структура однофайлової програми

Однофайлова програма мовою C++ являє собою послідовність препроцесорних директив, описів і визначень глобальних об'єктів і функцій (див. рис. 1.5).

Препроцесорні директиви управляють перетворенням тексту програми до її компіляції. Це рядки, що починаються із символу #, слідом за яким йде ім'я директиви (наприклад, #define або #include) і додаткова інформація. Препроцесор виконує пошук у тексті програми рядків, що починаються із символу #, і перетворення тексту програми відповідно до вмісту цього рядка. Препроцесорні директиви можуть перебувати в будь-якому місці програми.

Найчастіше за допомогою препоцесорних директив здійснюється підключення бібліотек для розширення функціональних можливостей коду. Бібліотека – це набір функцій (у тому числі зі стандартних бібліотек), а також визначених поза програмою змінних та констант, які можуть бути використані в програмі й зберігаються у відкомпільованому вигляді.

Отриманий у результаті роботи препроцесора повний текст програми обробляється компілятором, що створює об'єктний файл – текст програми машинною мовою, який не може бути виконаний через відсутність зв'язку з компонентами, що перебувають у інших файлах. Виконуваний модуль (exe-файл) будується з об'єктних файлів (obj-файлів) компонувальником, що збирає відкомпільований текст програми і функції з бібліотек в одну виконувану програму.

Визначення вводять функції та об'єкти, а описи повідомляють компілятор про властивості й імена тих об'єктів і функцій, які визначені в інших частинах програми (нижче по тексту або в іншому файлі).

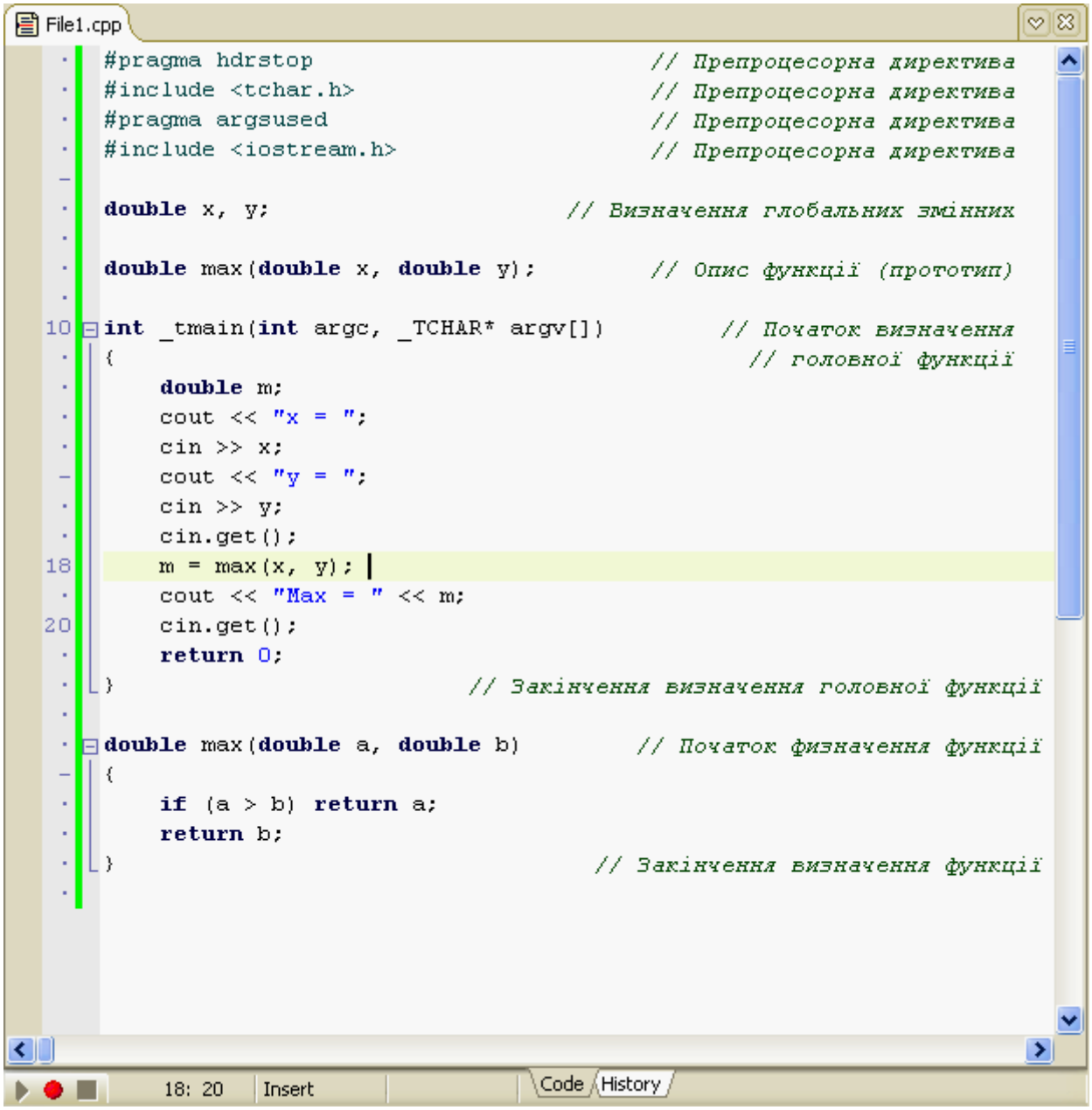

Рис. 1.5. Структура однофайлової програми

Функції визначають потенційно можливі дії програми і є самостійними одиницями. Кожна функція має ім’я та список аргументів, забраний в круглі дужки. Дужки вказуються навіть при відсутності аргументів, що дозволяє відрізнити ім’я функції від імен змінних. Далі у фігурних дужках записується тіло функції, що являє собою послідовність операторів, які у мові С++ завершуються символом «крапка з комою».

1.5. Налаштування програми

Розглянемо більш детально типи помилок і повідомлень та методи їх усунення.

Процес усунення помилок у програмі називається ***налаштуванням*** (debugging), а самі помилки в програмі досить часто називають ***жучками*** (bugs). Серед помилок, які допускаються в програмі, виділяють синтаксичні помилки, помилки часу виконання і логічні помилки.

Синтаксичні помилки – це помилки, пов'язані з порушенням ***синтаксису*** (тобто правил граматики) мови програмування. Прикладом такої помилки, що часто зустрічається у програмах, є пропуск символу «крапка з комою», яким завершується кожен оператор у програмах, написаних мовою C++. При виявленні синтаксичної помилки компілятор видає ***повідомлення про помилку***, вказуючи її передбачуване місце розташування і пояснюючи можливий її зміст. Звісно, природа помилки може відрізнятися від тієї її інтерпретації, яку робить компілятор, тим більше що одна синтаксична помилка може призвести до того, що компілятор видасть декілька повідомлень про помилки, зумовлені, проте, наявністю лише першої з них. Може бути і протилежний випадок, коли одна синтаксична помилка ховає від компілятора інші помилки, що є далі у тексті програми.

Досить часто компілятор виводить ***попереджувальні повідомлення*** про деякі дещо незвичні конструкції у програмі. У багатьох випадках програмісти звертають увагу тільки на повідомлення компілятора про наявність помилок (Errors), не реагуючи на попередження (Warnings). У той же час попереджувальні повідомлення можуть свідчити про помилки, у зв'язку з чим на них варто обов'язково звертати увагу і вносити відповідні виправлення. В ідеальному випадку попереджувальні повідомлення відсутні.

За умовчанням виведення попереджень компілятора передбачене не для усіх випадків. Для того щоб попередження виводилися на екран, треба виконати команду Projects ►Options... (комбінація клавіш Shift+Ctrl+F11), вибрати у відкриваному при цьому вікні пункт C++ Compiler ►Warnings і встановити значення True у рядку Enabled all warnings.

Для запуску процесу компіляції програми слід обрати пункт меню Project ►Build (гаряча клавіша Ctrl+F9). Якщо створений програмний код є некоректним, то по закінченню процесу компіляції на екран буде відображено

вікно (див. рис. 1.6), в якому міститься сумарна інформація про виявлені в процесі компіляції помилки (Errors) та попередження (Warnings).

Можливим є також вибір пункту меню Run ▶Run (гаряча клавіша F9 або клік мишкою над інструментальною кнопкою). В останньому випадку, насправді, мова йде про запуск програми на виконання. Але перед виконанням програми, якщо у її тексті здійснювалася корекція, будуть автоматично виконані компіляція програми та редагування зв'язків. У разі відсутності синтаксичних помилок програма буде автоматично запущена, інакше їх необхідно усунути. Якщо ж здійснювалася тільки компіляція і проект був зібраний без помилок, то кінцевий вигляд вікна компіляції буде аналогічний тому, що зображений на рис. 1.7.

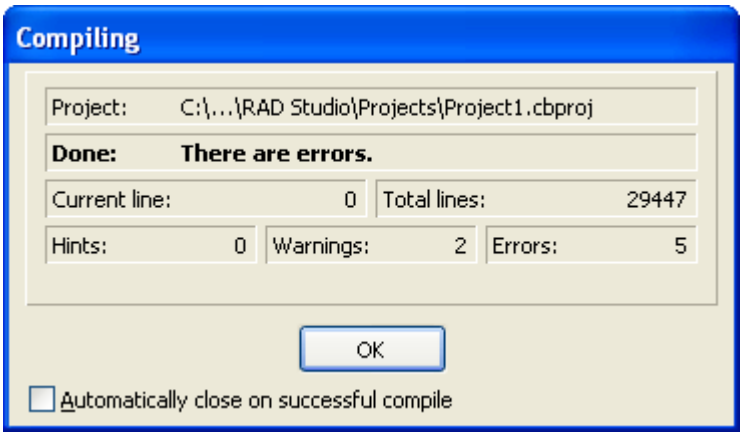

Рис 1.6. Вікно компіляції проекту у разі наявності помилок

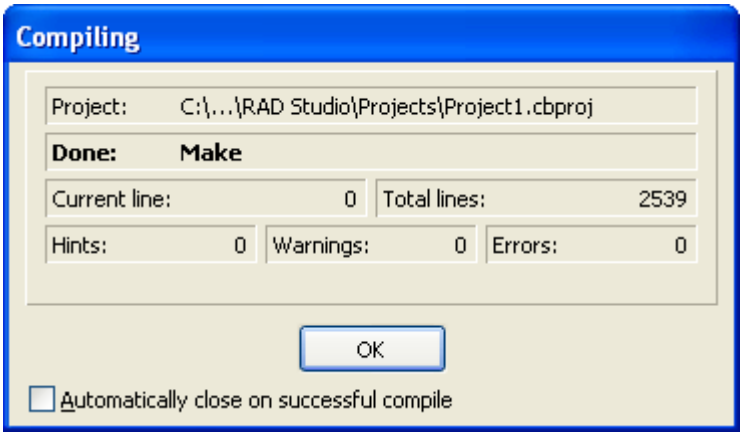

Рис 1.7. Вікно компіляції проекту у разі відсутності помилок

Деякі помилки проявляються тільки під час виконання програми, у зв'язку з чим вони носять відповідну назву – ***помилки часу виконання*** (run-

time errors). У термінології C++Builder такі помилки називаються винятками (Exceptions). З появою таких помилок програма завершується аварійно з видачею пояснювального повідомлення. У середовищі поява таких помилок призводить до переривання виконання програми і виведення вікна з інформацією про вид виключення (рис. 1.8).

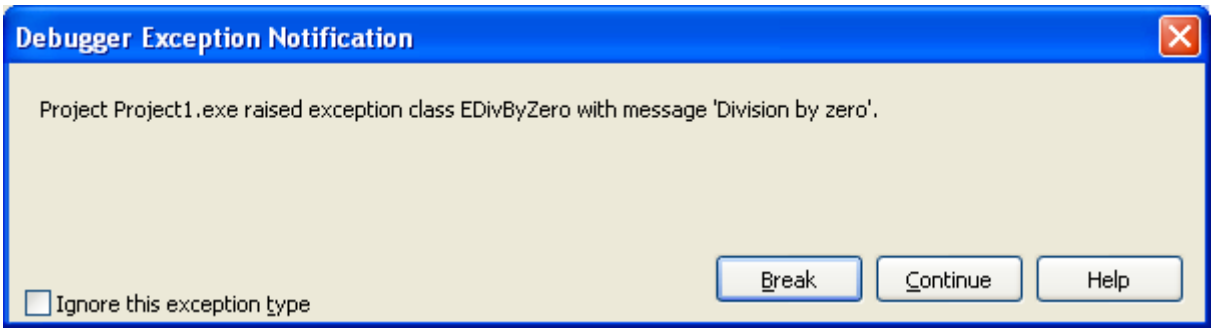

Рис. 1.8. Вікно виникнення виключення

Прикладами таких помилок часу виконання є ділення на нуль, одержання великих числових значень, які не можуть бути записані в комірку пам'яті (переповнення).

Часто найбільш неприємними помилками є ***логічні помилки*** – помилки в самому алгоритмі, а також помилки, спричинені елементарною неуважністю (наприклад, використання в програмі операції * замість операції +, задавання неправильного числового значення, використання одного імені змінної замість іншого). Такі помилки компілятор найчастіше виявити не може (за винятком випадків, коли вони призводять до порушення синтаксису). Мало того, логічні помилки можуть не проявити себе і під час виконання програми як помилки часу виконання.

Для виявлення логічних помилок здійснюється ***тестування*** програми, яке виражається в її запуску з кількома характерними наборами вхідних даних і перевірці відповідних їм результатів. При цьому в жодному разі не можна обмежуватися одноразовою перевіркою програми – повинні бути відслідковані всі окремі випадки вхідних даних, з якими програма може зіткнутися. Тільки після перевірки правильності функціонування програми на багатьох характерних наборах даних можна отримати високу (але не абсолютну) гарантію її правильності.

Найчастіше за все для того, щоб знайти причину помилки, треба виконати якийсь фрагмент програми, спостерігаючи значення змінних при виконанні кожної команди.

Для виявлення більшості логічних помилок передбачений спеціальний засіб, що зветься налагоджувальником (Debugger).

Робота в режимі налагоджування – це, насамперед, виконання програми по кроках з відслідковуванням послідовності виконання операторів програми та значень, які набувають змінні після виконання того або іншого кроку.

Для виконання фрагменту програми по кроках можна використовувати такі команди меню:

- Run ►Trace Into (Трасування із заходом у ...) – покрокове виконання програми із заходом у функції з наступним покроковим виконанням їх рядків (аналогом є натискання клавіші F7 або клік мишкою над інструментальною кнопкою );
- Run ►Step Over (По крокам без заходу у ...) – покрокове виконання рядків програми, при якому виклик функції вважається за одну команду, тобто вхід у функції не проводиться (аналогом є натискання клавіші F8 або клік мишкою над інструментальною кнопкою );
- Run ►Trace to Next Source Line (Трасування до наступного рядка, Shift+F7) – перехід до наступного рядка програми (аналогом є натискання клавіші Shift+F7);
- Run ►Run to Cursor (Виконати до курсору, F4) – команда виконує програму до того виконуваного оператора, на якому розташований курсор у вікні редактора коду (аналогом є натискання клавіші F4);
- Run Until Return (Виконати до виходу з функції) – виконання програми до виходу з поточної функції і зупинка на операторі, наступному за викликом цієї функції, з залишенням курсору у рядку, в якому було здійснене звертання до функції (аналогом є натискання клавіші Shift+F8 або клік мишкою над інструментальною кнопкою );
- Run ►Show Execution Point (Показати точку виконання) – команда показує на екрані виконуваний рядок коду та переміщує на нього курсор.

Досить часто для налаштування програми використовуються так звані точки переривання. Щоб ввести просту (безумовну) точку переривання,

достатньо у вікні редактора коду клікнути мишкою на лівій межі вікна навпроти необхідної рядка коду (або ж помістити курсор у необхідний рядок і натиснути клавішу F5). При цьому здійсниться забарвлення рядка, а навпроти нього на лівій межі вікна коду з'явиться червона кулька. Якщо тепер запустити програму на виконання, то кожен раз коли керування перейде до рядка, в якому вказана точка переривання, відбувається переривання виконання.

Дія точки переривання дещо аналогічна натисканню клавіші F4, але перевага точок переривання полягає в тому, що можна одночасно вказати декілька таких точок в різних місцях коду і в різних модулях. Програма в цьому разі виконується до першої точки переривання, яка зустрінеться під час виконання. Після аналізу проміжних результатів можна продовжити виконання програми, натиснувши інструментальну кнопку  .

Точки переривання можна встановлювати тільки на виконуваних операторах. Прибрати точку переривання можна, виконавши ті ж дії, що виконувалися для її встановлення.

Для перегляду значення змінних у момент зупинки програми досить навести курсор мишки на ім'я необхідної змінної в коді програми. Якщо ж є необхідність відслідковувати стан одразу декількох змінних, їх можна додати в вікно перегляду Watches List (див. рис 1.9).

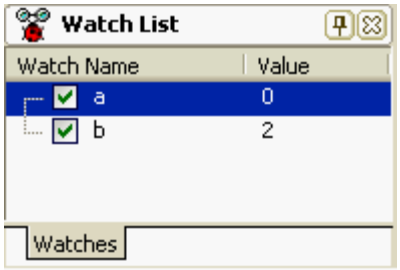

Рис. 1.9. Вікно перегляду стану змінних

За умовчанням вікно пергляду стану змінних автоматично виводиться у режимі налаштування ліворуч від вікна коду. У разі відсутності потреби у цьому вікні, й можна закрити. Щоб викликати раніше закрите вікно перегляду стану змінних, необхідно вибрати в головному меню опцію View ► Debug Windows ► Watches або ж натиснути комбінацію клавіш Ctrl+Alt+W.

Для додавання змінних у вікно Watches List достатньо здійснити подвійний клік мишкою в цьому вікні і ввести назву змінної в рядок Expression

вікна, яке відкриється після кліку. Якщо у вікні коду здійснене виділення імені деякої змінної, то для додавання цієї змінної до списку Watches List досить натиснути Ctrl+F5. Для видалення імені з вікна Watches List треба активізувати його у вікні та натиснути клавішу Insert або скористатися правою кнопкою миші і вибрати відповідний пункт у контекстному меню, що відкривається за кліком правої кнопки мишки.

За умовчанням у режимі налаштування ліворуч від вікна коду під вікном Watches List виводиться вікно локальних змінних Local Variables (див. рис. 1.10).

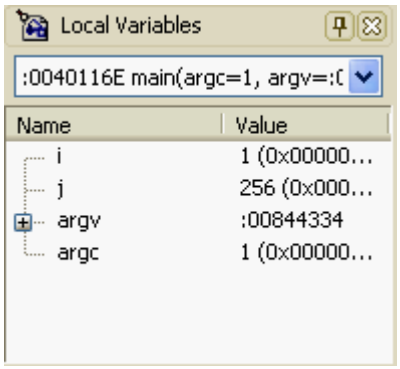

Рис. 1.10. Вікно локальних змінних

У цьому вікні виводяться значення локальних змінних функцій (у тому числі їх формальних параметрів). У разі закриття вікна локальних змінних, його повторне відкриття здійснюється вибором в головному меню опції View ► Debug Windows ► Local Variables або ж натисканням комбінації клавіш Ctrl+Alt+L.

Якщо вміст вікна локальних змінних визначається автоматично, то вміст вікна Watches List визначається програмістом.

2. ПРИКЛАД СТВОРЕННЯ НОВОГО КОНСОЛЬНОГО ЗАСТОСУВАННЯ ТА ЙОГО НАЛАШТУВАННЯ

Розглянемо можливу послідовність дій по налаштуванню програми на прикладі задачі розробки програми відшукання коренів квадратного рівняння

$$ax^2 + bx + c = 0,\ a \neq 0.$$

Далі умовно окремі групи дій будемо нумерувати з метою їх структуризації.

Етап 1. Наберемо такий текст програми:

```
#include <tchar.h>
#include <iostream.h>
#include <math.h>
//---------------------------------------------------------
#pragma argsused
int _tmain(int argc, _TCHAR* argv[])
{
    double a, b, c, x1, x2;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> a;
    cin.get();
    d = b * b - 4 * a * a;
    if (d < 0)
      cout << "The equation has no roots";
    cout << "   x1 = " << (-b + sqrt(d)) / 2 * a
    cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
    cin.get();
    return 0;
}
//---------------------------------------------------------
```

Етап 2. Забезпечимо виведення попереджувальних повідомлень компілятора, та виконаємо компіляцію. Як видно з рис. 1.11, компілятор виявив 3 синтаксичні помилки та 3 попередження і вивів відповідні пояснення у спеціальне вікно Messages. Не будемо поки звертати увагу на попередження і спробуємо виправити помилки.

Закриємо вікно компіляції, клікнувши мишкою над кнопкою Ok у ньому. Після цього стане активним вікно коду, у якому забарвиться червоним кольором рядок, у якому вперше була знайдена помилка, а курсор розміститься за яким-небудь символом цього рядка. Програміст повинен шукати синтаксичну помилку перед курсором за текстом програми, причому не обов'язково у цьому ж рядку. Якщо здійснити подвійний клік мишкою у вікні Messages над будь-яким з рядків, у вікні коду буде здійснений перехід до рядка, у якому вперше знайдена відповідна помилка.

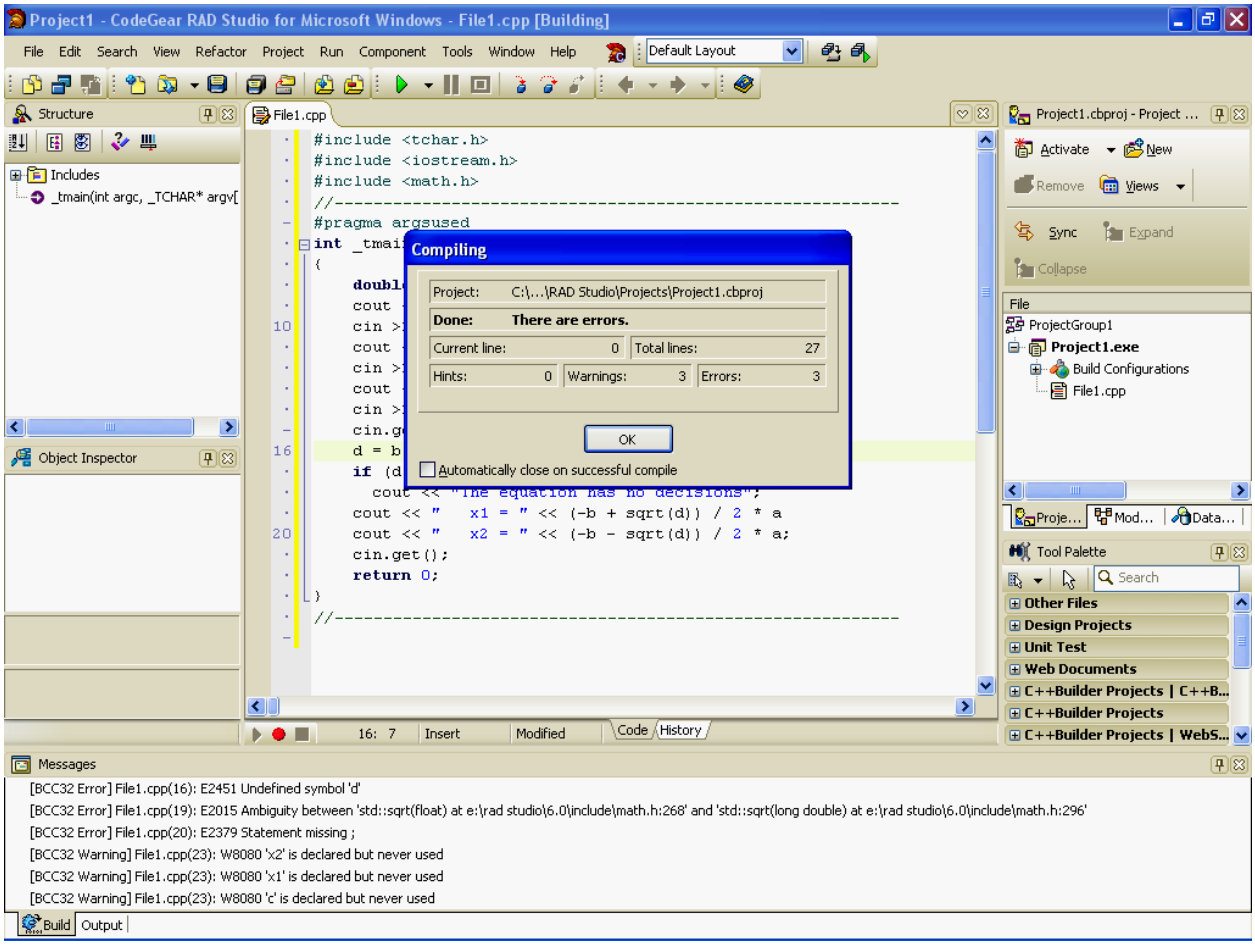

Рис. 1.11. Результати компіляції

На рис 1.11 у вікні Messages виведена інформація про такі три помилки:

[BCC32 Error] File1.cpp(20): E2451 Undefined symbol 'd' – Невизначений символ 'd';

[BCC32 Error] File1.cpp(23): E2015 Ambiguity between 'std::sqrt(float) at e:\rad studio\6.0\include\math.h:268' and 'std::sqrt(long double) at e:\rad studio\6.0\include\math.h:296' – Двозначність між 'std::sqrt(float) at e:\rad studio\6.0\include\math.h:268' та 'std::sqrt(long double) у e:\rad studio\6.0\include\math.h:296;

[BCC32 Error] File1.cpp(24): E2379 Statement missing ; – Відсутній символ ;.

Перше повідомлення говорить про те, що змінна d (для запису значення дискримінанту) використовується, але не визначена у програмі. Тому включимо її у перелік змінних, описуваних у верхній частині тексту програми.

Дещо складнішою є друга помилка. Справа у тому, що у C++ можна використовувати функції, що мають одне й те саме ім’я і відрізняються параметрами. У даному разі йдеться про використання стандартної функції об-

числення кореня квадратного sqrt(), яка може мати, як параметр типу **float**, так і параметр типу **long double**. Оскільки змінна d не визначена, виникає невизначеність, яку виявив компілятор. Судячи за все друга помилка усувається, якщо правильно визначити тип змінної d, усуваючи першу помилку, тобто має місце два повідомлення, обумовлені однією помилкою.

Відзначимо, що обидві помилки виправлялися не у тих рядках, де вони були виявлені, хоча змінну d у цьому випадку можна було визначити і у рядку, в якому була виявлена перша помилка, приписавши перед її іменем тип **double**.

Третя помилка вказує на те, що оператори мови повинні закінчуватися символом «крапка з комою». Місце де виявлена помилка не співпадає з місцем, де вона була зроблена – треба дописати символ «крапка з комою» у кінець попереднього рядка, завершивши записаний у ньому оператор.

Таким чином, ми виявили не три, а дві помилки, і виправлений текст набирає такий вигляд:

```
#include <iostream.h>
#include <math.h>
//---------------------------------------------------------
#pragma argsused
int _tmain(int argc, _TCHAR* argv[])
{
    double a, b, c, d, x1, x2;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> a;
    cin.get();
    d = b * b - 4 * a * a;
    if (d < 0)
      cout << "The equation has no roots";
    cout << "   x1 = " << (-b + sqrt(d)) / 2 * a;
    cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
    cin.get();
    return 0;
}
//---------------------------------------------------------
```

Етап 3. Натиснемо інструментальну кнопку  , запускаючи програму на виконання, і переконаємось, що компіляція пройде успішно і програма почне виконуватися, видавши повідомлення про необхідність уведення коефіцієнту *a*. Якщо ввести значення *a* = 1, *b* = 2, *c* = 1, то ми переконаємося, що отримані корені, є правильними. Однак для *a* = 1, *b* = 5, *c* = 2 будуть отримані неправильні корені. Тому треба вже шукати логічні помилки.

Етап 4. Відкриємо вікно Watches List, додамо в нього змінну d, помістимо курсор у рядок, наступний за обчисленням значення дискримінанта d, і натиснемо клавішу F4. Після введення вказаних вище значень коефіцієнтів переконуємося, що дискримінант обчислений невірно (у вікні Watches List виведено значення d, що дорівнює 9, замість 17). Помічаємо, що у тексті програми оператор

```
d = b * b - 4 * a * a;
```

неправильний. Виправляємо помилку:

```
d = b * b - 4 * a * c;
```

Якщо виконати ті ж дії, що й раніше, то ми переконаємося, що у вікні Watches List для змінної d буде виведене +NAN, яким позначається невизначене значення. Таким чином десь вище є логічна помилка, яку треба виправити.

Етап 5. Не перериваючи виконання програми, додамо у вікно Watches List змінні a, b, c. Бачимо, що змінна c має значення +NAN, хоч уведення її здійснювалося. Для відшукання помилки, припиняємо виконання програми і натискаємо клавішу F8 для нового запуску програми у режимі налаштування. Після декількох натискань клавіші F8 і введення вказаних трьох значень, ми помічаємо, що третій оператор уведення замість того, щоб записати значення 1 у замінну c, записав його у змінну a (це можна було побачити дещо раніше, але людина звичайно бачить ті значення, що явно виділяються – тут значення +NAN у змінній c).

Вказану помилку можна було побачити ще після першої компіляції. Річ у тому, що тоді ми не звернули увагу на три попередження, в яких вказувалося, що змінні x1, x2 та c оголошені, але ніколи не використовуються. Більш того, виправивши синтаксичні помилки, ми одразу запустили програму на виконання, не виконавши попередньої компіляції. Якби попередня компіляція була виконана без запуску програми на виконання, ми б могли

побачити, що попереджувальне повідомлення відносно змінної c змінилося, набравши такий вигляд:

[BCC32 Warning] File1.cpp(21): W8013 Possible use of 'c' before definition – можливе використання змінної c перед визначенням.

Це повідомлення свідчить, що в операторі використовується змінна c, у яку явно не записувалося ніяке значення.

Таким чином, ми переконалися, що навіть попередження можуть свідчити про помилку і на них треба звертати увагу.

В той же час на повідомлення про змінні x1, x2 можна не звертати уваги, хоча краще видалити ці змінні з програми, виключивши тим самим появу попередження.

Таким чином, треба виправити другий оператор

```
cin >> a;
```

записавши його так:

```
cin >> c;
```

Якщо тепер запустити програму на виконання, то буде отриманий правильний результат, який однак ще не свідчить про відсутність логічних помилок, оскільки ми ще не перевіряли випадок відсутності коренів.

Етап 6. Запустимо програму на виконання і введемо тепер такі значення: $a = 1$, $b = 2$, $c = 3$. У цьому разі ми бачимо, що, з одного боку, програма виводить повідомлення про відсутність коренів, а, з другого боку, вона ще виводить повідомлення про наявність помилки часу виконання, а також два корені, значеннями яких є +NAN. Знов треба зайнятися налаштуванням.

Натискаючи декілька разів клавішу F8, бачимо, що оператори виконуються у потрібній послідовності (при цьому будуть по черзі забарвлюватися виконувані рядки), але після виведення повідомлення про відсутність коренів, здійснюється перехід до рядка, у якому обчислюється та виводиться перший корінь, хоч цей рядок не повинен виконуватися. Справа в тому, що у програмі використаний невірний для цієї задачі формат оператора **if**.

Вставимо перед рядком, де виводиться перший корінь рівняння, такий рядок:

```
else
```

Якщо тепер запустити програму на виконання, то ми побачимо, що помилка залишилася, але тільки по відношенню до другого кореня.

За допомогою клавіші F8 переконуємося у тому, що у програмі виконується один оператор уведення, виконання якого не повинне було мати місце. Справа у тому, що у цьому випадку два оператори

```
cout << "   x1 = " << (-b + sqrt(d)) / 2 * a;
cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
```

повинні розглядатися як один так званий складений оператор, для чого їх треба укласти у фігурні дужки.

Зробимо це і отримаємо такий текст програми:

```
#include <iostream.h>
#include <math.h>
//---------------------------------------------------------
#pragma argsused
int _tmain(int argc, _TCHAR* argv[])
{
    double a, b, c, d;
    cout << "a = ";
    cin >> a;
    cout << "b = ";
    cin >> b;
    cout << "c = ";
    cin >> c;
    cin.get();
    d = b * b - 4 * a * c;
    if (d < 0)
      cout << "The equation has no roots";
    else {
      cout << "   x1 = " << (-b + sqrt(d)) / 2 * a;
      cout << "   x2 = " << (-b - sqrt(d)) / 2 * a;
    }
    cin.get();
    return 0;
}
```

Запуск програми зі значеннями $a = 1$, $b = 2$, $c = 3$ тепер приведе до отримання правильного результату.

Етап 7. Запустимо програму на виконання і введемо нові значення коефіцієнтів: $a = 2$, $b = 5$, $c = 3$. У цьому разі ми бачимо, що обчислені корені (`x1 = -4`, `x2 = -6`) знову є неправильними (повинні бути отримані значення `x1 = -1`, `x2 = -1.5`). Треба знову здійснювати покрокове виконан-

ня програми, натискаючи клавішу F8. Оскільки ми не передбачили (точніше, знищили) змінні для запису коренів рівняння, прийдеться дивитися на результат, що виводиться на екран користувача. В результаті ми побачимо, що всі обчислення до виведення значення першого кореня виконуються правильно, а сам корінь є невірним. Висновок – є помилка у обчисленні кореня. Уважно подивившись на оператор, ми бачимо, що в ньому порушений порядок виконання операцій, а саме, оскільки операції множення та ділення мають один і той же пріоритет, при обчисленні кореня спочатку здійснюється ділення на 2, після чого отриманий результат помножується на a. Щоб отримати правильний результат, необхідно забрати знаменник `2 * a` у дужки, причому це, звісно, потрібно зробити і при обчисленні другого кореня.

Після виправлення тексту отримуємо правильний результат.

Таким чином, одноразове отримання правильного результату не дає гарантії того, що програма буде правильно. Остання помилка не проявлялася раніше, тому що для коефіцієнта *a* уводилося значення 1, яке не впливало на результат і у разі попадання цього значення у чисельник, і у разі попадання його у знаменник. Це ще раз говорить про необхідність ретельного підбору тестових даних та багаторазового тестування.

3. ЗАВДАННЯ НА ЛАБОРАТОРНУ РОБОТУ

Під час лабораторної роботи потрібно:

1. Набрати текст програми, наведений на початку розділу 2, і повторити дії, пов’язані з налаштуванням програми.

2. Виконати наведені нижче завдання, пов’язані з набиранням тексту програми для нового консольного застосування, збереженням його у файлі, збереженням з новим іменем, завантаженням тексту програми з диска, редагуванням та налаштуванням програми.

Завдання 1.

Створіть консольне застосування, в якому повторіть наведений нижче текст програми. З’ясуйте, що виконує програма.

```
#include <iostream.h>
int _tmain(int argc, _TCHAR* argv[])
{
    int a, b, c;
    cin >> a >> b;
    if (a > b)
```

```
        cout << "1 - Yes" << endl;
    else
        cout << "1 - No" << endl;
    c = a * b;
    if (c > 100)
        cout << "2 - Yes" << endl;
    else
        cout << "2 - No" << endl;
    if (a % 2 == b % 2)
        cout << "3 - Yes" << endl;
    else
        cout << "3 - No" << endl;
    return 0;
}
```

Завдання 2.

Нижче наведено код програми з синтаксичними помилками. На основі цього коду створіть консольний додаток, в якому усуньте всі знайдені синтаксичні помилки.

```
#include <iostream.h>
int _tmain(int argc, _TCHAR* argv[])
{
    int a;
    cin >> a;
    cin >> b;
    if (a <> 0 && b <> 0)
    {
        if (a * b > 0)
            cout << 'Yes' << endl;
        else
            cout << 'No' << endl
    else
        cout << '?' << endl;
}.
```

Завдання 3.

Нижче наведено варіант коду програми, що нормує вектор. У коді містяться помилки. На основі наведеного коду створіть коректно працююче консольне застосування.

```
#include <iostream.h>
#include <math.h>
```

```
int _tmain(int argc, _TCHAR* argv[])
{
    double a, b, c, d;
    cin >> a >> a;
    c = sqrt(a * a + b * b);
    a = a / d;
    b = b / d;
    cout << a << "\t" << b;
}
```

4. КОНТРОЛЬНІ ЗАПИТАННЯ

1. Які дії треба виконати для створення нового консольного застосування?
2. Як виконується налаштування вікна New Console Application?
3. Як здійснюється збереження вмісту вікна коду?
4. Як здійснюється збереження вмісту вікна коду у файлі з новим іменем?
5. Як записати на диск проект разом із файлом коду консольного застосування?
6. Опишіть дії, пов’язані з керуванням курсором у вікні редактора.
7. Як можна здійснювати видалення символів у тексті редакторського вікна? Опишіть дію клавіш видалення.
8. Як здійснюється виділення тексту?
9. Як можна скопіювати фрагмент тексту в інше місце?
10. Як можна перенести фрагмент тексту в інше місце?
11. Що являє собою однофайлова програма, написана мовою C++?
12. Яке призначення компілятора?
13. Чи правда, що компілятор оброблює вихідний текст?
14. Як поділяються помилки у програмі? Опишіть їх особливості.
15. Що створює компілятор у разі відсутності синтаксичних помилок?
16. Яке призначення компонувальника (редактора зв’язків) і що є результатом його роботи?
17. Яке призначення препроцесорних директив і як вони записуються?
18. Що відслідковується при покроковому виконанні програми?
19. У чому полягає відмінність режимів покрокового виконання Trace Into та Step Over?
20. Що таке «виконання до курсору»?

21. Що таке точка переривання і як вона встановлюється та видаляється?
22. У чому відмінність точки переривання і виконання до курсору?
23. Чи можна одночасно встановити декілька точок переривання?
24. Для чого використовується вікно Watches List?
25. Як внести нове ім’я у вікно Watches List?
26. Як видалити ім’я з вікна Watches List?

СПИСОК ЛІТЕРАТУРИ

1. Страуструп, Б. Язык программирования Си++ : Второе издание / Б. Страуструп. – К. : ДиаСофт, 1993. – Ч. 1. – 264 с. ; Ч. 2. – 296 с.
2. Керниган, Б. Язык программирования Си / Б. Керниган, Д. Ритчи. – М. : Финансы и статистика, 1992. – 272 с.
3. Либерти, Джесс. Освой самостоятельно C++ за 21 день : учеб. пособ. / Джесс Либерти. – М. : Издательский дом «Вильямс», 2001. – 816 с.
4. Подбельский, В. В. Программирование на языке Си / В. В. Подбельский, С. С. Фомин. – М. : Финансы и статистика, 1999. – 600 с.
5. Подбельский, В. В. Язык Си++ / В. В. Подбельский. – М. : Финансы и статистика, 1999. – 560 с.
6. Савитч, Уолтер. Язык C++. Курс объектно-ориентированного программирования / Уолтер Савитч. – М. : Издательский дом «Вильямс», 2001. – 704 с.

ЗМІСТ

1. Теоретичні основи 3
   1.1. Початкові дії 3
   1.2. Робота з вікном коду 5
   1.3. Компіляція та компонування 9
   1.4. Структура однофайлової програми 10
   1.5. Налаштування програми 12
2. Приклад створення нового консольного застосування та його налаштування 17
3. Завдання на лабораторну роботу 24
4. Контрольні запитання 26

Список літератури 27

Навчальне видання

Методичні вказівки
до лабораторної роботи
«Створення консольного застосування у C++Builder 2009»
з курсу «Програмування» для студентів напряму 6.040302 – Інформатика
і курсу «Програмування та алгоритмічні мови» для студентів напряму
6.040303 – Системний аналіз

Укладачі: БЕЗМЕНОВ Микола Іванович,
МАРЧЕНКО Ігор Іванович

Відповідальний за випуск О. С. Куценко
Роботу до видання рекомендував О. В. Горелий

За авторською редакцією

План 2010 р., поз. 7 / 37-10

Підп. до друку 15.03.2010 р. Формат 60 × 84 $^{1}/_{16}$. Папір офісний.
Riso-друк. Гарнітура Таймс. Ум. друк. арк. 1,6. Наклад 50 прим.
Зам. № 58. Ціна договірна.

Видавничий центр НТУ «ХПІ».
Свідоцтво про державну реєстрацію ДК № 3657 від 24.12.2009 р.
61002, Харків, вул. Фрунзе, 21

Друкарня НТУ «ХПІ», 61002, Харків, вул. Фрунзе, 21